WEST Search History

Hide Items | Restore | Clear | Cancel

DATE: Wednesday, February 25, 2004

Hide?	Set Name	Query	Hit Count
	DB=PGPB,USPT,USOC; PLUR=YES; OP=ADJ		
☐	L15	L10 and partition$	6
	DB=DWPI,TDBD; PLUR=YES; OP=ADJ		
☐	L14	L13 and restructur$	1
☐	L13	L6	148
	DB=EPAB; PLUR=YES; OP=ADJ		
☐	L12	L11 and basic block	0
☐	L11	L6	9
	DB=PGPB,USPT,USOC; PLUR=YES; OP=ADJ		
☐	L10	L9 and (profile or profiling)	10
☐	L9	L8 and basic block	19
☐	L8	L4 and (l1 or l2)	58
	DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=ADJ		
☐	L7	L3 and l6	4
☐	L6	node near3 weight	1272
☐	L5	(node and edge) same weight	715
☐	L4	(node or edge) same weight	218276
☐	L3	(code or instruction) near2 restructur$	152
	DB=PGPB,USPT,USOC; PLUR=YES; OP=ADJ		
☐	L2	711/117-129,153,170-173,207,208.ccls.	5770
☐	L1	717/136,137,140-147,149-161.ccls.	2029

END OF SEARCH HISTORY

Search Results

Search Results for: **[profile<AND>(((code restructuring)))]**
Found **30** of **127,132 searched.**

Search within Results

profiling Go > Advanced Search > Search Help/Tips

Sort by: **Title** **Publication** **Publication Date** **Score** Binder

Results 1 - 20 of 30 **short listing**

Prev Page 1 2 Next Page

1 Interprocedural conditional branch elimination 87%

Rastislav Bodík , Rajiv Gupta , Mary Lou Soffa

ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 1997 conference on Programming language design and implementation May 1997

Volume 32 Issue 5

The existence of statically detectable correlation among conditional branches enables their elimination, an optimization that has a number of benefits. This paper presents techniques to determine whether an interprocedural execution path leading to a conditional branch exists along which the branch outcome is known at compile time, and then to eliminate the branch along this path through code restructuring. The technique consists of a demand driven interprocedural analysis that determines whethe ...

2 Optimized code restructuring of OS/2 executables 83%

Jyh-Herng Chow , Yong-fong Lee , Kalyan Muthukumar , Vivek Sarkar , Mauricio Serrano , Iris Garcia , John Hsu , Shauchi Ong , Honesty Young

Proceedings of the 1995 conference of the Centre for Advanced Studies on Collaborative research November 1995

This paper describes the design and algorithms of FDPR/2 (Feedback Directed Program Restructuring of OS/2 executables), a general-purpose tool that can be used to instrument, profile, and restructure/optimize OS/2 executables for the tel x86 architecture. The optimizations delivered by FDPR/2's restructuring include improved utilization of the (instruction) memory hierarchy, improved branch alignment, and dead code elimination. These optimizations are known to be critical for object-oriented pro ...

3 Compiler support for block buffering 82%

Mahmut Kandemir , J. Ramanujam , Ugur Sezer

Proceedings of the 2001 international symposium on Low power electronics and design August 2001

4 Metric (Extended Abstract): A kernel instrumentation system for distributed environments 80%

Gene Mcdaniel

Proceedings of the sixth ACM symposium on Operating systems principles November 1977

Metric is a distributed software measurement system that communicates measurement data over the PARC computer network, the Ethernet. Metric is used to instrument stand alone and distributed computer systems (it works in an environment of about 90 machines total and is used by about 15 machines). The system is divided into three parts: object system probes that transmit measurement events, the accountant that receives and stores those events, and the analyst that manipulates the d ...

5 Online feedback-directed optimization of Java 80%

Matthew Arnold , Michael Hind , Barbara G. Ryder

ACM SIGPLAN Notices , Proceedings of the 17th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications November 2002 Volume 37 Issue 11

This paper describes the implementation of an online feedback-directed optimization system. The system is fully automatic; it requires no prior (offline) profiling run. It uses a previously developed low-overhead instrumentation sampling framework to collect control flow graph edge profiles. This profile information is used to drive several traditional optimizations, as well as a novel algorithm for performing feedback-directed control flow graph node splitting. We empirically evaluate this syst ...

6 Synthesis for Low Power: Source code transformation based on software cost analysis 80%

Eui-Young Chung , Luca Benini , Giovanni De Micheli

Proceedings of the 14th international symposium on Systems synthesis September 2001

This paper presents a model and a strategy for source-code transformation applied to software application programs to reduce their energy cost. We propose a flexible performance and energy model for a processor-memory system. The benefit of the model is generality (it is not tied to a single memory and processor architecture) and effectiveness of evaluation. With this model, we first estimate the effects of source-code transformations (called transformation cost), representing the improvement ra ...

7 Partial dead code elimination using slicing transformations 80%

Rastislav Bodík , Rajiv Gupta

ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 1997 conference on Programming language design and implementation May 1997 Volume 32 Issue 5

We present an approach for optimizing programs that uncovers additional opportunities for optimization of a statement by *predicating* the statement. In this paper predication algorithms for achieving partial dead code elimination (PDE) are presented. The process of predication embeds a statement in a control flow structure such that the statement is executed only if the execution follows a path along which the value computed by the statement is live. The control flow restructuring performe ...

8 Research track: Improving spatial locality of programs via data mining 77%

Karlton Sequeira , Mohammed Zaki , Boleslaw Szymanski , Christopher Carothers

Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining August 2003

In most computer systems, page fault rate is currently minimized by generic page

replacement algorithms which try to model the temporal locality inherent in programs. In this paper, we propose two algorithms, one greedy and the other stochastic, designed for program specific code restructuring as a means of increasing spatial locality within a program. Both algorithms effectively decrease average working set size and hence the page fault rate. Our methods are more effective than traditional appr ...

9 Object equality profiling 77%

Darko Marinov , Robert O'Callahan

ACM SIGPLAN Notices , Proceedings of the 18th ACM SIGPLAN conference on Object-oriented programing, systems, languages, and applications October 2003

Volume 38 Issue 11

We present *Object Equality Profiling* (OEP), a new technique for helping programmers discover optimization opportunities in programs. OEP discovers opportunities for replacing a set of equivalent object instances with a single representative object. Such a set represents an opportunity for automatically or manually applying optimizations such as hash consing, heap compression, lazy allocation, object caching, invariant hoisting, and more. To evaluate OEP, we implemented a tool to help prog ...

10 Microprocessor architecture: A scalable wide-issue clustered VLIW with a reconfigurable interconnect 77%

Osvaldo Colavin , Davide Rizzo

Proceedings of the international conference on Compilers, architectures and synthesis for embedded systems October 2003

Clustered VLIW architectures have been widely adopted in modern embedded multimedia applications for their ability to exploit high degrees of ILP with reasonable trade-off in complexity and silicon costs. Studies have however shown limited performance scaling for wide-issue machines. In this paper we describe the architecture of a clustered VLIW with a runtime reconfigurable inter-cluster bus suitable to address such scalability problem. The architecture is aimed at kernel loops acceleration thr ...

11 Achieving high instruction cache performance with an optimizing compiler 77%

W. W. Hwu , P. P. Chang

ACM SIGARCH Computer Architecture News , Proceedings of the 16th annual international symposium on Computer architecture April 1989

Volume 17 Issue 3

Increasing the execution power requires a high instruction issue bandwidth, and decreasing instruction encoding and applying some code improving techniques cause code expansion. Therefore, the instruction memory hierarchy performance has become an important factor of the system performance. An instruction placement algorithm has been implemented in the IMPACT-I (Illinois Microarchitecture Project using Advanced Compiler Technology - Stage I) C compiler to maximize the sequential and spatial ...

12 A comparison of automatic parallelization tools/compilers on the SGI origin 2000 77%

Michael Frumkin , Michelle Hribar , Haoqiang Jin , Abdul Waheed , Jerry Yan

Proceedings of the 1998 ACM/IEEE conference on Supercomputing (CDROM) November 1998

Porting applications to new high performance parallel and distributed computing platforms is a challenging task. Since writing parallel code by hand is time consuming and costly, porting codes would ideally be automated by using some parallelization tools and compilers. In this paper, we compare the performance of three parallelization tools and compilers based on the NAS Parallel Benchmark and a CFD application, ARC3D, on the SGI Origin2000 multiprocessor. The tools and compilers compared inclu ...

Search Results

Search Results for: **[partition<AND>((profile<AND>(((code restructuring)))))]**
Found **7** of **127,132 searched.**

Search within Results

1 Research track: Improving spatial locality of programs via data mining 77%

Karlton Sequeira , Mohammed Zaki , Boleslaw Szymanski , Christopher Carothers

Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining August 2003

In most computer systems, page fault rate is currently minimized by generic page replacement algorithms which try to model the temporal locality inherent in programs. In this paper, we propose two algorithms, one greedy and the other stochastic, designed for program specific code restructuring as a means of increasing spatial locality within a program. Both algorithms effectively decrease average working set size and hence the page fault rate. Our methods are more effective than traditional appr ...

2 Object equality profiling 77%

Darko Marinov , Robert O'Callahan

ACM SIGPLAN Notices , Proceedings of the 18th ACM SIGPLAN conference on Object-oriented programing, systems, languages, and applications October 2003 Volume 38 Issue 11

We present *Object Equality Profiling* (OEP), a new technique for helping programmers discover optimization opportunities in programs. OEP discovers opportunities for replacing a set of equivalent object instances with a single representative object. Such a set represents an opportunity for automatically or manually applying optimizations such as hash consing, heap compression, lazy allocation, object caching, invariant hoisting, and more. To evaluate OEP, we implemented a tool to help prog ...

3 A comparison of automatic parallelization tools/compilers on the SGI origin 2000 77%

Michael Frumkin , Michelle Hribar , Haoqiang Jin , Abdul Waheed , Jerry Yan

Proceedings of the 1998 ACM/IEEE conference on Supercomputing (CDROM) November 1998

Porting applications to new high performance parallel and distributed computing platforms is a challenging task. Since writing parallel code by hand is time consuming and costly, porting codes would ideally be automated by using some parallelization tools and compilers. In this paper, we compare the performance of three parallelization tools and compilers based on the

NAS Parallel Benchmark and a CFD application, ARC3D, on the SGI Origin2000 multiprocessor. The tools and compilers compared inclu ...

4 Load-reuse analysis: design and evaluation 77%

Rastislav Bodík , Rajiv Gupta , Mary Lou Soffa

ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 1999 conference on Programming language design and implementation May 1999

Volume 34 Issue 5

Load-reuse analysis finds instructions that repeatedly access the same memory location. This location can be promoted to a register, eliminating redundant loads by reusing the results of prior memory accesses. This paper develops a load-reuse analysis and designs a method for evaluating its precision.In designing the analysis, we aspire for *completeness*---the goal of exposing all reuse that can be harvested by a subsequent program transformation. For register promotion, a suitable transfo ...

5 Register promotion by sparse partial redundancy elimination of loads and stores 77%

Raymond Lo , Fred Chow , Robert Kennedy , Shin-Ming Liu , Peng Tu

ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 1998 conference on Programming language design and implementation May 1998

Volume 33 Issue 5

An algorithm for register promotion is presented based on the observation that the circumstances for promoting a memory location's value to register coincide with situations where the program exhibits partial redundancy between accesses to the memory location. The recent SSAPRE algorithm for eliminating partial redundancy using a sparse SSA representation forms the foundation for the present algorithm to eliminate redundancy among memory accesses, enabling us to achieve both computational and li ...

6 Complete removal of redundant expressions 77%

Rastislav Bodík , Rajiv Gupta , Mary Lou Soffa

ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 1998 conference on Programming language design and implementation May 1998

Volume 33 Issue 5

Partial redundancy elimination (PRE), the most important component of global optimizers, generalizes the removal of common subexpressions and loop-invariant computations. Because existing PRE implementations are based on *code motion*, they fail to completely remove the redundancies. In fact, we observed that 73% of loop-invariant statements cannot be eliminated from loops by code motion alone. In dynamic terms, traditional PRE eliminates only half of redundancies that are strictly partial. ...

7 The design of a new frontal code for solving sparse, unsymmetric systems 77%

I. S. Duff , J. A. Scott

ACM Transactions on Mathematical Software (TOMS) March 1996

Volume 22 Issue 1

We describe the design, implementation, and performance of a frontal code for the solution of large, sparse, unsymmetric systems of linear equations. The resulting software package, MA42, is included in Release 11 of the Harwell Subroutine Library and is intended to supersede the earlier MA32 package. We discuss in detail the extensive use of higher-level BLAS kernels within MA42 and illustrate the performance on a range of practical problems on a CRAY Y-MP, an IBM 3090, and an IBM RISC Sys ...

Results 1 - 7 of 7 **short listing**

13 The UCLA AGCM in high performance computing environments 77%

C. R. Mechoso , L. A. Drummond , J. D. Farrara , J. A. Spahr

Proceedings of the 1998 ACM/IEEE conference on Supercomputing (CDROM) November 1998

General Circulation Models (GCMs) are at the top of the hierarchy of numerical models that are used to study the Earth's climate. To increase the significance of predictions using GCMs requires ensembles of integrations that in turn demand large amounts of computing resources. GCMs codes are particularly difficult to optimize in view of their heterogeneity. In this paper we focus on code optimization for GCMs of the atmosphere (AGCMs), one of the major components of the climate system.In this pa ...

14 Perfect Benchmarks™ decomposition and performance on VAX™ multiprocessors 77%

Zarka Cvetanovic , Edward G. Freedman , Charles Nofsinger

Proceedings of the 1990 ACM/IEEE conference on Supercomputing November 1990

This paper presents the methods for decomposition and the performance of the Perfect Benchmark suite on two VAX multiprocessors. Our results indicate that by several efficient decomposition techniques we were able to obtain significant performance gains on both VAX multiprocessors relative to a uniprocessor case. We propose a methodology that can be applied for decomposing other existing scientific and engineering applications and provide guidelines for auto-decomposing compiler improvements. Fi ...

15 Implementing product line variabilities 77%

Critina Gacek , Michalis Anastasopoules

ACM SIGSOFT Software Engineering Notes , Proceedings of the 2001 symposium on Software reusability: putting software reuse in context May 2001

Volume 26 Issue 3

Software product lines have numerous members. Thus, a product line infrastructure must cover various systems. This is the significant difference to usual software systems and the reason for additional requirements on the various assets present during software product line engineering. It is imperative that they support the description of the product line as a whole, as well as its instantiation for the derivation of individual products. Literature has already addressed how to cre ...

16 Control flow optimization for supercomputer scalar processing 77%

Pohua P. Chang , Wen-mei W. Hwu

Proceedings of the 3rd international conference on Supercomputing June 1986

Control intensive scalar programs pose a very different challenge to highly pipelined supercomputers than vectorizable numeric applications. Function call/return and branch instructions disrupt the flow of instructions through the pipeline, degrading the utilization of the pipelined datapaths. This paper describes control flow optimization for scalar processing using an optimizing compiler. To obtain program control flow information, a system independent profiler has been integrated into th ...

17 Load-reuse analysis: design and evaluation 77%

Rastislav Bodík , Rajiv Gupta , Mary Lou Soffa

ACM SIGPLAN Notices , Proceedings of the ACM SIGPLAN 1999 conference on Programming language design and implementation May 1999

Volume 34 Issue 5

Load-reuse analysis finds instructions that repeatedly access the same memory location. This location can be promoted to a register, eliminating redundant loads by reusing the results of prior memory accesses. This paper develops a load-reuse analysis and designs a

method for evaluating its precision.In designing the analysis, we aspire for *completeness*--- the goal of exposing all reuse that can be harvested by a subsequent program transformation. For register promotion, a suitable transfo ...

18 A flexible code generation framework for the design of application specific programmable processors 77%
François Charot , Vincent Messé
Proceedings of the seventh international workshop on Hardware/software codesign March 1999

19 A graphic parallelizing environment for user-compiler interaction 77%
C. R. Calidonna , M. Giordano , M. Mango Furnari
Proceedings of the 13th international conference on Supercomputing May 1999

20 Improving the performance of speculatively parallel applications on the Hydra CMP 77%
Kunle Olukotun , Lance Hammond , Mark Willey
Proceedings of the 13th international conference on Supercomputing May 1999

Results 1 - 20 of 30 **short listing**

Prev Page 1 2 Next Page

Hit List

Clear | Generate Collection | Print | Fwd Refs | Bkwd Refs | Generate OACS

Search Results - Record(s) 1 through 4 of 4 returned.

☐ 1. Document ID: US 6381739 B1

L7: Entry 1 of 4 File: USPT Apr 30, 2002

US-PAT-NO: 6381739
DOCUMENT-IDENTIFIER: US 6381739 B1
** See image for Certificate of Correction **

TITLE: Method and apparatus for hierarchical restructuring of computer code

DATE-ISSUED: April 30, 2002

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Breternitz, Jr.; Mauricio	Austin	TX		
Smith; Roger A.	Austin	TX		

US-CL-CURRENT: 714/37; 714/38

ABSTRACT:

A compiler (142) constructs (FIGS. 14-32) a Reduced Flowgraph (RFG) from computer source code (144). The RFG is used to instrument (FIG. 36) code (142). An object module is created (146) and executed (148). Resulting path frequency counts are written to a counts file (154). A compiler (158) uses the source code (144) and the generated counts to identify runtime correlations between successive path edges and Superedges. An object module (159) is generated containing reordered (156) code generated to optimize performance based on the runtime correlations. If cloning is enabled (152), high frequency path edges are cloned (154) or duplicated to minimize cross edge branching.

51 Claims, 43 Drawing figures
Exemplary Claim Number: 1
Number of Drawing Sheets: 28

Full | Title | Citation | Front | Review | Classification | Date | Reference | Claims | KWMC | Draw Desc | In

☐ 2. Document ID: US 6175957 B1

L7: Entry 2 of 4 File: USPT Jan 16, 2001

US-PAT-NO: 6175957
DOCUMENT-IDENTIFIER: US 6175957 B1

TITLE: Method of, system for, and computer program product for providing efficient utilization of memory hierarchy through code restructuring

DATE-ISSUED: January 16, 2001

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Ju; Dz Ching	Sunnyvale	CA		
Muthukumar; Kalyan	Cupertino	CA		
Ramaswamy; Shankar	Bethel Park	PA		
Simons; Barbara Bluestein	Palo Alto	CA		

US-CL-CURRENT: 717/156; 717/159

ABSTRACT:

Code restructuring or reordering based on profiling information and memory hierarchy is provided by constructing a Program Execution Graph (PEG) corresponding to a level of the memory hierarchy, partitioning this PEG to reduce estimated memory overhead costs below an upper bound, and constructing a PEG for a next level of the memory hierarchy from the partitioned PEG. The PEG is constructed from control flow and frequency information from a profile of the program to be restructured. The PEG is a weighted undirected graph comprising nodes representing basic blocks and edges representing transfer of control between pairs of basic blocks. The weight of a node is the size of the basic block it represents and the weight of an edge is the frequency of transition between the pair of basic blocks it connects. The nodes of the PEG are partitioned or clustered into clusters such that the sum of the weights of the nodes in any cluster is no greater than an upper bound. A next PEG is then constructed from the clusters of the partitioned PEG such that a node in the next PEG corresponds to a cluster in the partitioned PEG, and such that there is an edge between two nodes in the next PEG if there is an edge between the clusters represented by the two nodes. Weights are assigned to the nodes and edges of the next PEG to produce a PEG, and then the PEG partitioning, basic block reordering, and PEG construction steps may be repeated for each level of the memory hierarchy. After the clustering is completed, the basic blocks are reordered in memory by grouping all of the nodes of a cluster in an adjacent order beginning at a boundary for all the levels of the memory hierarchy. Because clusters must not cross boundaries of memory hierarchies, NOPs are added to fill out the portion of a memory hierarchy level that is not filled by the clusters.

15 Claims, 8 Drawing figures
Exemplary Claim Number: 1
Number of Drawing Sheets: 8

☐ 3. Document ID: US 5889999 A

L7: Entry 3 of 4 File: USPT Mar 30, 1999

US-PAT-NO: 5889999
DOCUMENT-IDENTIFIER: US 5889999 A

TITLE: Method and apparatus for sequencing computer instruction execution in a data processing system

DATE-ISSUED: March 30, 1999

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Breternitz, Jr.; Mauricio	Austin	TX		
Smith; Roger A.	Austin	TX		

US-CL-CURRENT: 717/158; 712/201, 712/214, 712/233, 712/236, 712/237, 714/35, 714/45

ABSTRACT:

A method and apparatus for sequencing computer instructions in memory (24) to provide for more instruction efficient execution by a central processing unit (CPU) (22) begins by executing the computer instructions via the CPU (22) and creating a trace file (FIG. 2) in memory (24). The trace file is then scanned using a window size greater than two (i.e., more than two instructions or basic blocks/ groups of instructions are selected as each window) and correlations are determined between several pairs of instructions in each window (FIGS. 9 and 10). The correlations obtained by the window procedure are then analyzed (FIG. 11) to determine an efficient ordering of computer instructions for subsequent execution by any target CPU.

33 Claims, 43 Drawing figures
Exemplary Claim Number: 1
Number of Drawing Sheets: 28

☐ 4. Document ID: US 5742814 A

L7: Entry 4 of 4 File: USPT Apr 21, 1998

US-PAT-NO: 5742814
DOCUMENT-IDENTIFIER: US 5742814 A

TITLE: Background memory allocation for multi-dimensional signal processing

DATE-ISSUED: April 21, 1998

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Balasa; Florin	Tustin Ranch	CA		
Catthoor; Francky	Temse			BE
De Man; Hugo	Kessel-lo			BE

US-CL-CURRENT: 707/102; 707/104.1, 717/155

ABSTRACT:

Data storage and transfer cost is responsible for a large amount of the VLSI system realization cost in terms of area and power consumption for real-time multi-dimensional signal processing applications. Applications or this type are data-dominated because they handle a large amount of indexed data which are produced and consumed in the

context of nested loops. This important application domain includes the majority of speech, video, image, and graphic processing (multi-media in general) and end-user telecom applications. The present invention relates to the automated allocation of the background memory units, necessary to store the large multi-dimensional signals. In order to handle both procedural and nonprocedural specification, the novel memory allocation methodology is based on an optimization process driven by data-flow analysis. This steering mechanism allows more exploration freedom than the more restricted scheduling-based investigation in the existent synthesis systems. Moreover, by means of an original polyhedral model of data-flow analysis, the novel allocation methodology can accurately deal with complex specifications, containing large multi-dimensional signals. The class of specifications handled by this polyhedral model covers a larger range than the conventional ones, i.e. the entire class of affine representations. Employing estimated silicon area or power consumption costs yielded by recent models for on-chip memories, the novel allocation methodology produces one, or optionally, several distributed multi-port memory architecture(s) with fully-determined characteristics, complying with a given clock cycle budget for memory operations.

46 Claims, 50 Drawing figures
Exemplary Claim Number: 1
Number of Drawing Sheets: 41

Terms	Documents
L3 and L6	4

Hit List

Clear | Generate Collection | Print | Fwd Refs | Bkwd Refs | Generate OACS

Search Results - Record(s) 1 through 4 of 4 returned.

☐ 1. Document ID: US 6381739 B1

L7: Entry 1 of 4 File: USPT Apr 30, 2002

US-PAT-NO: 6381739
DOCUMENT-IDENTIFIER: US 6381739 B1
**** See image for Certificate of Correction ****

TITLE: Method and apparatus for hierarchical restructuring of computer code

DATE-ISSUED: April 30, 2002

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Breternitz, Jr.; Mauricio	Austin	TX		
Smith; Roger A.	Austin	TX		

US-CL-CURRENT: 714/37; 714/38

ABSTRACT:

A compiler (142) constructs (FIGS. 14-32) a Reduced Flowgraph (RFG) from computer source code (144). The RFG is used to instrument (FIG. 36) code (142). An object module is created (146) and executed (148). Resulting path frequency counts are written to a counts file (154). A compiler (158) uses the source code (144) and the generated counts to identify runtime correlations between successive path edges and Superedges. An object module (159) is generated containing reordered (156) code generated to optimize performance based on the runtime correlations. If cloning is enabled (152), high frequency path edges are cloned (154) or duplicated to minimize cross edge branching.

51 Claims, 43 Drawing figures
Exemplary Claim Number: 1
Number of Drawing Sheets: 28

Full | Title | Citation | Front | Review | Classification | Date | Reference | ... | Claims | KWMC | Draw. Desc | Ir

☐ 2. Document ID: US 6175957 B1

L7: Entry 2 of 4 File: USPT Jan 16, 2001

US-PAT-NO: 6175957
DOCUMENT-IDENTIFIER: US 6175957 B1

TITLE: Method of, system for, and computer program product for providing efficient utilization of memory hierarchy through code restructuring

DATE-ISSUED: January 16, 2001

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Ju; Dz Ching	Sunnyvale	CA		
Muthukumar; Kalyan	Cupertino	CA		
Ramaswamy; Shankar	Bethel Park	PA		
Simons; Barbara Bluestein	Palo Alto	CA		

US-CL-CURRENT: 717/156; 717/159

ABSTRACT:

Code restructuring or reordering based on profiling information and memory hierarchy is provided by constructing a Program Execution Graph (PEG) corresponding to a level of the memory hierarchy, partitioning this PEG to reduce estimated memory overhead costs below an upper bound, and constructing a PEG for a next level of the memory hierarchy from the partitioned PEG. The PEG is constructed from control flow and frequency information from a profile of the program to be restructured. The PEG is a weighted undirected graph comprising nodes representing basic blocks and edges representing transfer of control between pairs of basic blocks. The weight of a node is the size of the basic block it represents and the weight of an edge is the frequency of transition between the pair of basic blocks it connects. The nodes of the PEG are partitioned or clustered into clusters such that the sum of the weights of the nodes in any cluster is no greater than an upper bound. A next PEG is then constructed from the clusters of the partitioned PEG such that a node in the next PEG corresponds to a cluster in the partitioned PEG, and such that there is an edge between two nodes in the next PEG if there is an edge between the clusters represented by the two nodes. Weights are assigned to the nodes and edges of the next PEG to produce a PEG, and then the PEG partitioning, basic block reordering, and PEG construction steps may be repeated for each level of the memory hierarchy. After the clustering is completed, the basic blocks are reordered in memory by grouping all of the nodes of a cluster in an adjacent order beginning at a boundary for all the levels of the memory hierarchy. Because clusters must not cross boundaries of memory hierarchies, NOPs are added to fill out the portion of a memory hierarchy level that is not filled by the clusters.

15 Claims, 8 Drawing figures
Exemplary Claim Number: 1
Number of Drawing Sheets: 8

☐ 3. Document ID: US 5889999 A

L7: Entry 3 of 4 File: USPT Mar 30, 1999

US-PAT-NO: 5889999
DOCUMENT-IDENTIFIER: US 5889999 A

TITLE: Method and apparatus for sequencing computer instruction execution in a data processing system

DATE-ISSUED: March 30, 1999

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Breternitz, Jr.; Mauricio	Austin	TX		
Smith; Roger A.	Austin	TX		

US-CL-CURRENT: 717/158; 712/201, 712/214, 712/233, 712/236, 712/237, 714/35, 714/45

ABSTRACT:

A method and apparatus for sequencing computer instructions in memory (24) to provide for more instruction efficient execution by a central processing unit (CPU) (22) begins by executing the computer instructions via the CPU (22) and creating a trace file (FIG. 2) in memory (24). The trace file is then scanned using a window size greater than two (i.e., more than two instructions or basic blocks/ groups of instructions are selected as each window) and correlations are determined between several pairs of instructions in each window (FIGS. 9 and 10). The correlations obtained by the window procedure are then analyzed (FIG. 11) to determine an efficient ordering of computer instructions for subsequent execution by any target CPU.

33 Claims, 43 Drawing figures
Exemplary Claim Number: 1
Number of Drawing Sheets: 28

☐ 4. Document ID: US 5742814 A

L7: Entry 4 of 4 File: USPT Apr 21, 1998

US-PAT-NO: 5742814
DOCUMENT-IDENTIFIER: US 5742814 A

TITLE: Background memory allocation for multi-dimensional signal processing

DATE-ISSUED: April 21, 1998

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Balasa; Florin	Tustin Ranch	CA		
Catthoor; Francky	Temse			BE
De Man; Hugo	Kessel-lo			BE

US-CL-CURRENT: 707/102; 707/104.1, 717/155

ABSTRACT:

Data storage and transfer cost is responsible for a large amount of the VLSI system realization cost in terms of area and power consumption for real-time multi-dimensional signal processing applications. Applications or this type are data-dominated because they handle a large amount of indexed data which are produced and consumed in the

context of nested loops. This important application domain includes the majority of speech, video, image, and graphic processing (multi-media in general) and end-user telecom applications. The present invention relates to the automated allocation of the background memory units, necessary to store the large multi-dimensional signals. In order to handle both procedural and nonprocedural specification, the novel memory allocation methodology is based on an optimization process driven by data-flow analysis. This steering mechanism allows more exploration freedom than the more restricted scheduling-based investigation in the existent synthesis systems. Moreover, by means of an original polyhedral model of data-flow analysis, the novel allocation methodology can accurately deal with complex specifications, containing large multi-dimensional signals. The class of specifications handled by this polyhedral model covers a larger range than the conventional ones, i.e. the entire class of affine representations. Employing estimated silicon area or power consumption costs yielded by recent models for on-chip memories, the novel allocation methodology produces one, or optionally, several distributed multi-port memory architecture(s) with fully-determined characteristics, complying with a given clock cycle budget for memory operations.

46 Claims, 50 Drawing figures
Exemplary Claim Number: 1
Number of Drawing Sheets: 41

Terms	Documents
L3 and L6	4

Hit List

Clear | Generate Collection | Print | Fwd Refs | Bkwd Refs | Generate OACS

Search Results - Record(s) 1 through 6 of 6 returned.

☐ 1. Document ID: US 6175957 B1

L15: Entry 1 of 6 File: USPT Jan 16, 2001

US-PAT-NO: 6175957
DOCUMENT-IDENTIFIER: US 6175957 B1

TITLE: Method of, system for, and computer program product for providing efficient utilization of memory hierarchy through code restructuring

DATE-ISSUED: January 16, 2001

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Ju; Dz Ching	Sunnyvale	CA		
Muthukumar; Kalyan	Cupertino	CA		
Ramaswamy; Shankar	Bethel Park	PA		
Simons; Barbara Bluestein	Palo Alto	CA		

US-CL-CURRENT: 717/156; 717/159

ABSTRACT:

Code restructuring or reordering based on profiling information and memory hierarchy is provided by constructing a Program Execution Graph (PEG) corresponding to a level of the memory hierarchy, partitioning this PEG to reduce estimated memory overhead costs below an upper bound, and constructing a PEG for a next level of the memory hierarchy from the partitioned PEG. The PEG is constructed from control flow and frequency information from a profile of the program to be restructured. The PEG is a weighted undirected graph comprising nodes representing basic blocks and edges representing transfer of control between pairs of basic blocks. The weight of a node is the size of the basic block it represents and the weight of an edge is the frequency of transition between the pair of basic blocks it connects. The nodes of the PEG are partitioned or clustered into clusters such that the sum of the weights of the nodes in any cluster is no greater than an upper bound. A next PEG is then constructed from the clusters of the partitioned PEG such that a node in the next PEG corresponds to a cluster in the partitioned PEG, and such that there is an edge between two nodes in the next PEG if there is an edge between the clusters represented by the two nodes. Weights are assigned to the nodes and edges of the next PEG to produce a PEG, and then the PEG partitioning, basic block reordering, and PEG construction steps may be repeated for each level of the memory hierarchy. After the clustering is completed, the basic blocks are reordered in memory by grouping all of the nodes of a cluster in an adjacent order beginning at a boundary for all the levels of the memory hierarchy. Because clusters must not cross boundaries of memory hierarchies, NOPs are added to fill out the portion of a memory hierarchy level that is not filled by the clusters.

15 Claims, 8 Drawing figures
Exemplary Claim Number: 1
Number of Drawing Sheets: 8

Full | Title | Citation | Front | Review | Classification | Date | Reference | Claims | KWIC | Draw. Desc | Ir

---

☐ 2. Document ID: US 5963972 A

L15: Entry 2 of 6 File: USPT Oct 5, 1999

US-PAT-NO: 5963972
DOCUMENT-IDENTIFIER: US 5963972 A

TITLE: Memory architecture dependent program mapping

DATE-ISSUED: October 5, 1999

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Calder; Bradley Gene	Palo Alto	CA		
Hashemi; Amir Hooshang	Brighton	MA		
Kaeli; David Richard	Medway	MA		

US-CL-CURRENT: 711/129; 711/118, 711/133, 711/158

ABSTRACT:

In a computer implemented method, instructions of a program are mapped into a cache memory of a computer system. The cache memory is partitioned into a plurality fixed size lines for the convenience of accessing the instructions. Each block is assigned a different identification, for example a unique color. The program is partitioned into a plurality of instruction units, for example procedures or basic blocks. A flow graph is generated for the program. In the graph, nodes represent the instructions units, and edges directly connect nodes that have an execution relationship. Instruction units of directly connected nodes are mapped into blocks having different identifications or colors. An unavailable-set of identifications is maintained for each node. The unavailability-set of a particular node includes the identifications of blocks mapping instruction units directly connected to the particular node and which should not be used for the particular procedure in order to minimize cache conflicts during execution of the program.

15 Claims, 9 Drawing figures
Exemplary Claim Number: 1
Number of Drawing Sheets: 9

Full | Title | Citation | Front | Review | Classification | Date | Reference | Claims | KWIC | Draw. Desc | Ir

---

☐ 3. Document ID: US 5887174 A

L15: Entry 3 of 6 File: USPT Mar 23, 1999

US-PAT-NO: 5887174
DOCUMENT-IDENTIFIER: US 5887174 A

TITLE: System, method, and program product for instruction scheduling in the presence of hardware lookahead accomplished by the rescheduling of idle slots

DATE-ISSUED: March 23, 1999

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Simons; Barbara Bluestein	Palo Alto	CA		
Sarkar; Vivek	Palo Alto	CA		

US-CL-CURRENT: 717/161; 712/216, 713/502, 717/156

ABSTRACT:

Instructions are scheduled for execution by a processor having a lookahead buffer by identifying an idle slot in a first instruction schedule of a first basic block of instructions, and by rescheduling the idle slot later in the first instruction schedule. The idle slot is rescheduled by determining if the first basic block of instructions may be rescheduled into a second instruction schedule in which the identified idle slot is scheduled later than in the first instruction schedule. The first basic block of instructions is rescheduled by determining a completion deadline of the first instruction schedule, decreasing the completion deadline, and determining the second instruction schedule based on the decreased completion deadline. Deadlines are determined by computing a rank of each node of a DAG corresponding to the first basic block of instructions; constructing an ordered list of the DAG nodes in nondecreasing rank order; and applying a greedy scheduling heuristic to the ordered list. An instruction in a second subsequent basic block of instructions may be rescheduled to execute in the rescheduled idle slot. This process may be repeated for each idle slot.

18 Claims, 15 Drawing figures
Exemplary Claim Number: 1
Number of Drawing Sheets: 10

☐ 4. Document ID: US 5557761 A

L15: Entry 4 of 6 File: USPT Sep 17, 1996

US-PAT-NO: 5557761
DOCUMENT-IDENTIFIER: US 5557761 A

TITLE: System and method of generating object code using aggregate instruction movement

DATE-ISSUED: September 17, 1996

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Chan; Sun C.	Fremont	CA		

Dehnert; James C.	Palo Alto	CA
Lo; Raymond W.	Sunnyvale	CA
Towle; Ross A.	San Francisco	CA

US-CL-CURRENT: 717/156; 717/144, 717/146

ABSTRACT:

A system and method of generating object code from an intermediate representation of source code is described. The intermediate representation includes a plurality of basic blocks each being represented by a plurality dam dependency graphs, wherein each data dependency graph comprises a plurality of nodes each corresponding to an instruction from the target computer instruction set. The present invention operates by selecting a source basic block (that is one of the basic blocks of the intermediate representation) and a target basic block (that is another of the basic blocks of the intermediate representation), and by identifying a maximal set of instructions contained in the source basic block that are movable from the source basic block to the target basic block without violating any data dependency relationships of the data dependency graphs. An overall cost model of aggregately moving instructions of the maximal set from the source basic block to the target basic block is generated. This cost model specifies an executable cost of moving each of the instructions of the maximal set from the source basic block to the target basic block. Then, the present invention aggregately moves one or more instructions of the maximal set from the source basic block to the target basic block according to the cost model to form the object code.

16 Claims, 37 Drawing figures
Exemplary Claim Number: 1
Number of Drawing Sheets: 18

☐ 5. Document ID: US 5530866 A

L15: Entry 5 of 6 File: USPT Jun 25, 1996

US-PAT-NO: 5530866
DOCUMENT-IDENTIFIER: US 5530866 A
**** See image for Certificate of Correction ****

TITLE: Register allocation methods having upward pass for determining and propagating variable usage information and downward pass for binding; both passes utilizing interference graphs via coloring

DATE-ISSUED: June 25, 1996

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Koblenz; Brian D.	Seattle	WA		
Callahan, II; Charles D.	Mercer Island	WA		

US-CL-CURRENT: 717/144; 717/158, 717/159

ABSTRACT:

The present invention provides methods for allocating physical registers within a compiler phase to achieve efficient operation of a target CPU. The methods of the present invention allocate variables between physical registers and memory to accommodate local as well as global code structure. Such methods facilitate the location of variables that are heavily accessed at a portion of the code in a physical register during the execution thereof.

34 Claims, 4 Drawing figures
Exemplary Claim Number: 1
Number of Drawing Sheets: 3

☐ 6. Document ID: US 5475842 A

L15: Entry 6 of 6 File: USPT Dec 12, 1995

US-PAT-NO: 5475842
DOCUMENT-IDENTIFIER: US 5475842 A

TITLE: Method of compilation optimization using an N-dimensional template for relocated and replicated alignment of arrays in data-parallel programs for reduced data communication during execution

DATE-ISSUED: December 12, 1995

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Gilbert; John R.	Palo Alto	CA		
Chatterjee; Siddhartha	Sunnyvale	CA		
Schreiber; Robert S.	Palo Alto	CA		

US-CL-CURRENT: 717/160

ABSTRACT:

When a data-parallel language like Fortran 90 is compiled for a distributed-memory machine, aggregate data objects (such as arrays) are distributed across the processor memories. The mapping determines the amount of residual communication needed to bring operands of parallel operations into alignment with each other. A common approach is to break the mapping into two stages: first, an alignment that maps all the objects to an abstract template, and then a distribution that maps the template to the processors. This disclosure deals with two facets of the problem of finding alignments that reduce residual communication; namely, alignments that vary in loops, and objects that permit of replicated alignments. It is shown that loop-dependent dynamic alignment is sometimes necessary for optimum performance, and algorithms are provided so that a compiler can determine good dynamic alignments for objects within "do" loops. Also situations are identified in which replicated alignment is either required by the program itself (via spread operations) or can be used to improve performance. An algorithm based on network flow is proposed for determing which objects to replicate so as to minimize the total amount of broadcast communication in replication.

1 Claims, 10 Drawing figures

Exemplary Claim Number: 1
Number of Drawing Sheets: 8

Full | Title | Citation | Front | Review | Classification | Date | Reference | Claims | KWIC | Draw Desc | In

Clear | Generate Collection | Print | Fwd Refs | Bkwd Refs | Generate OACS

Terms	Documents
L10 and partition$	6

Display Format: REV Change Format

Previous Page Next Page Go to Doc#

WEST Search History

Hide Items | Restore | Clear | Cancel

DATE: Wednesday, February 25, 2004

Hide?	Set Name	Query	Hit Count
	DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=ADJ		
☐	L5	(node and edge) same weight$	1040
☐	L4	(node and edge) same weight$c block	0
☐	L3	(Code or instruction) near2 restructur$	152
	DB=PGPB,USPT,USOC; PLUR=YES; OP=ADJ		
☐	L2	711/117-129,153,170-173,207-208.ccls.	5770
☐	L1	717/136-137,140-147,149-161.ccls.	2029

END OF SEARCH HISTORY